In Proceedings Ground System Architectures Workshop (GSAW 2003), March 4—6 2003, Manhattan Beach, CA USA.

An Extended Framework for the Validation and Verification of Situation-Aware Middleware Architectures

SangEun Kim[1], Peter In[2], and Ramesh Bharadwaj [3]

Abstract

Model checking is a technique that has been successfully applied for the validation and verification of hardware specifications and communication protocols. In mission-critical systems of NASA and the DoD, various mobile devices generate information dynamically, and their state changes with time. Most often, a situation serves as a trigger for a new situation. Therefore, it is necessary to extend existing model checking methods and tools in order to apply them for the validation and verification of situation-aware, mission-critical applications such as DoD command and control systems or the Navy's Total Ship Computing Environment. However, there are several problems to be overcome before these techniques become practical, such as overcoming the state explosion problem and adapting the V&V systems and algorithms to this application area.

In this paper, we propose a new technique, founded on combining existing techniques in theorem proving and model checking to extend the application area of existing pure model checking methods. This paper also introduces state space reduction methods based on abstraction that ameliorate the state explosion problem. Future distributed real-time and embedded system must necessarily be highly adaptable, secure, and reliable. Existing system development techniques therefore need to be extended so that future systems have the capability to meet these new system requirements.

1. Motivation

In mission-critical systems such as DoD Command and Control Systems (DCCS) or the Navy's future Total Ship Computing Environment (TSCE), it is foreseen that tens of thousands of dynamically configurable systems will collaborate to create and process information in a secure and adaptable manner. An emerging new technology that could be used to design and build such applications is known as Situation-Aware Middleware (SAM). This dynamically configurable middleware will include functionality to support information dissemination to users in a dependable, yet transparent manner. The middleware must provide an open, standards-compliant, ubiquitous communication channel between situation-aware applications and its architecture must be lightweight. However, since future systems will operate in a dynamic and situation changing environment, this increases the difficulty of validation and verification of such applications. In general, validation and verification (V&V) of current day software systems is very difficult because the number of reachable states of such systems is very large. Therefore, when applied to these systems, conventional tools fail to complete the verification task, and usually terminate prematurely after consuming all available system resources. Therefore, model checking has rarely been applied for the analysis of software specifications even though it has been successfully used for the validation and verification of device drivers and communication protocols. In the past, scalability was achieved by the use of efficient representation techniques, such as Binary Decision Diagrams (BDDs), that are a canonical representation for boolean formulae [1], to symbolically represent the state space of a system. However, this technique has limited applicability for software specifications, which include variables of richer data types than Boolean. For instance, common data types include enumerations, and integer and real types. In this paper, we propose an integrated validation and verification system, which we call the Extended Validation & Verification System (EVS), which is designed to overcomes some of the difficulties in applying model checking to situation-aware applications. EVS is based on two major ideas: One idea is to combine model checking with theorem proving methods to achieve scalability and ease-of-use. Another is to automatically apply property-driven abstraction methods in order to derive a simpler specification that will be amenable to automated analysis.

In the next section, we discuss details of the Situation Aware Middleware architecture used to develop situation-aware applications. In Section 3, we discuss EVS and its application to situation-aware applications, followed by the conclusion in Section 4.

2. Situation-Aware Middleware Architecture

The need for a conceptual architecture for Situation-Aware Middleware based on Reconfigurable Context-Sensitive Middleware (RCSM) is proposed in [2]. Ubiquitous applications need to be aware of several contexts in order to be able to adaptively communicate with each other across different network environments, such as mobile ad-hoc networks, the Internet, or even mobile phone networks. However, existing context-aware techniques often become inadequate in these applications where combinations of multiple contexts and users' actions need to be analyzed over a period of time. Situation-awareness in application software is considered a desirable feature in order to overcome this limitation. In addition to being context-sensitive, situation-aware applications can respond to both

[1] Dept. of Computer Science, Texas A&M University, College Station, TX 77840 USA, Tel: 1-979-845-5439, sangeunk@cs.tamu.edu
[2] Dept. of Computer Science, Texas A&M University, College Station, TX 77840 USA, Tel: 1-979-458-1547, hohin@cs.tamu.edu
[3] Center for High Assurance Computer Systems, Naval Research Laboratory, Washington DC, Tel: 1-202-767-7210, ramesh@itd.nrl.navy.mil

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **2003**	2. REPORT TYPE	3. DATES COVERED **00-00-2003 to 00-00-2003**
4. TITLE AND SUBTITLE **An Extended Framework for the Validation and Verification of Situation-Aware Middleware Architectures**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Center for High Assurance Computer Systems,4555 Overlook Avenue, SW,Washington,DC,20375**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES **The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**3**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

current and historical relationships between specific contexts and device-actions.

Figure 1 shows how all of RCSM's components are layered inside a device. The Object Request Broker of RCSM (R-ORB) assumes the availability of reliable transport protocols; one R-ORB per device usually suffices.

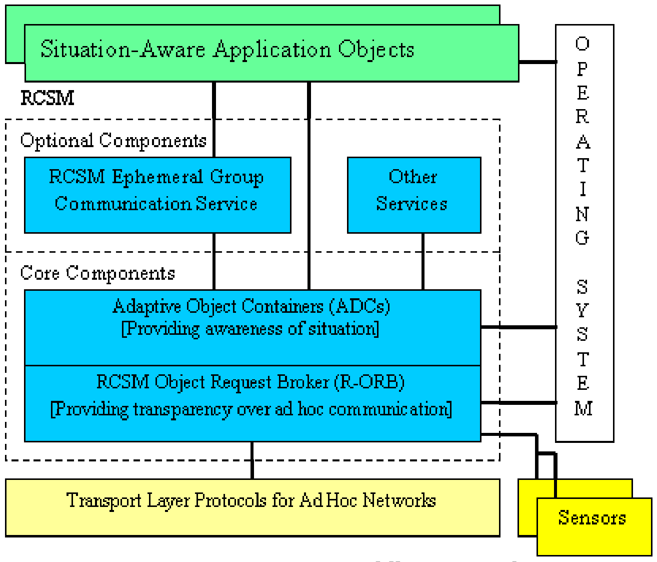
Situation-Aware Application Objects
RCSM
Optional Components
RCSM Ephemeral Group Communication Service
Other Services
Core Components
Adaptive Object Containers (ADCs) [Providing awareness of situation]
RCSM Object Request Broker (R-ORB) [Providing transparency over ad hoc communication]
OPERATING SYSTEM
Transport Layer Protocols for Ad Hoc Networks
Sensors

Figure 1. Situation-Aware Middleware Architecture

The number of ADCs (Adaptive Object Containers) depends on the number of context-sensitive objects in the device. ADCs periodically collect the necessary "raw context data" through the R-ORB, which in turn collects data from sensors and the operating system. Initially, each ADC registers with the R-ORB to express its needs for contexts and publishes the corresponding context-sensitive interface. RCSM is called re-configurable because it allows addition or deletion of individual ADCs during runtime (to manage new or existing context-sensitive application objects) without affecting other runtime operations inside RCSM. An example of a *SmartClassroom* is presented in [3].

3. Validation and Verification Framework

In Section 2, the situation-aware middleware architecture, which will be the basis for validation and verification, was introduced. In this section, we describe the architecture of the EVS, and the aspects of the EVS related to the proposed target middleware model with respect to the aspects of situation-awareness, real-time, and security.

During model checking, the state spaces of software systems are extremely large. Therefore, in current computing environments, applying model checking to such systems is often infeasible. Traditionally, there have been two main approaches to solve this problem, one using abstraction techniques and the other using partial order reduction methods. The idea of applying abstraction was first introduced by Clarke et al [4]. In this paper, the authors demonstrate how abstract finite state machines may be extracted directly from finite state programs, using a method similar to those used in abstract interpretation. The partial order reduction method consists of constructing a reduced state graph. The full state graph, which may be too big to fit in memory, need never be constructed [5]. This method is a good fit for asynchronous system verification.

The system EVS has two main capabilities, one to support abstraction mechanisms to reduce the state space and the other is to combine theorem proving and model checking. As shown in Figure 2, EVS consists of the following three sub-components:

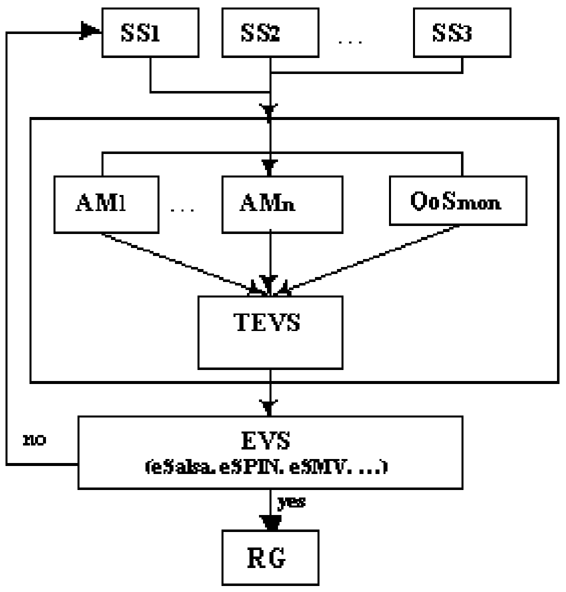

Figure 2: Our Validation and Verification Framework.
Legends:
SS: Situation Specification
AM: Abstraction Mechanism
QoSmon: QoS monitoring
TEVS: Translator EVS
EVS: Extended V&V System
RG: Report Generator

3.1 Abstraction Mechanisms Generating Component:

The inputs to this component include specifications of all the situations, in a formal notation called the situation-aware contract specification language, SA-CSL [6]. Conventional model checking and theorem proving tools are mainly used to specify and check safety properties. Therefore, in order to check that situation-awareness, real-time, and security Quality of Service (QoS) requirements are met by the middleware specified in SA-CSL, there is a need to extend their capabilities. Our proposed extensions can reduce redundant, unnecessary constraints related to situation-awareness, and real-time and security constraints.

These abstraction mechanisms make the system EVS (discussed in the sequel) to be more effective and efficient.

3.2 Extended Validation and Verification System:

The two major problems addressed by the system EVS are the state explosion problem during model checking and the need for user ingenuity and guidance in the case of theorem proving. A two-pronged approach is being used to tackle these issues: (1) Use of the invariant checker Salsa which uses a combination of theorem proving and model checking techniques to provide scalability and ease-of-use. (2) Automatic property-driven abstraction methods to transform a specification expressed in SA-CSL into a simpler specification that is amenable to automated analysis. Additionally, a compliance checker (CC) proves compliance of the specified situation-aware middleware and the real-time constraints, the security policies, and QoS requirements expected of the middleware. By ensuring that these properties are satisfied and that the middleware behaves as specified, we address the issue of *integrity* of the generated middleware. Generally, this approach of validation and verification requires formal definitions using a Kripke structure to model the situation-aware application. Let AP be a set of atomic propositions. A Kripke structure over AP is a four tuple K = {AP, S, A, L} where AP is a set of atomic propositions that hold in a specific system state, S is a finite set of situations, A ($A \subseteq S \times S$) is a finite set of actions and L ($S \rightarrow 2^{AP}$) is a labeling function that labels each situation with the set of atomic propositions that hold for that situation.

3.3 Translator for the Extended Validation and Verification System (TEVS):

The Translator for the Extended Validation and Verification System (TEVS) is an interface between the Abstraction Mechanism and EVS. We plan to develop the system EVS and its components by reusing existing components and tools. Therefore, the translator will serve as the “glue” code that will serve as the interface to this diversity of systems and notations.

4. Conclusion

In this paper, we describe the architecture and design of an extended V&V system for situation-aware middleware architectures for mission-critical applications. The benefits of using our framework are: (1) Ease of development of situation-awareness applications and (2) Ability to validate and verify them by the use of efficient techniques to deal with the state space explosion problem by the use of a combination of abstraction methods and invariant checking.

ACKNOWLEDGEMENTS
We would like to thank Dr. Stephen Yau, of Arizona State University, for the helpful discussions on this topic.

References

[1] Ramesh Bharadwaj and Connie Heitmeyer, “Model Checking Complete Requirements Specifications Using Abstraction,” Automated Software Engineering, Volume 6, pp. 37--68, 1999.
[2] S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," *IEEE Pervasive Computing,* 1(3), July-September 2002, pp. 33-40.
[3] R. Bharadwaj and S. Sims, “Salsa: Combining Constraint Solvers with BDDs for Automatic Invariant Checking”, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 00)*, March 2000, Germany.
[4] E. M. Clarke et al., “Model Checking and Abstraction,” ACM Transactions on Programming Languages and Systems, pp. 1512-1542, 16(5), September 1994.
[5] D. Peled, “Combining partial order reductions with on-the-fly model-checking,” Proceedings of the 1994 Workshop on Computer-Aided Verification, LNCS 818, pp. 377--390, Springer, 1994.
[6] S. S. Yau, Y. Wang, D. Huang, and H. In, “A Middleware Situation-Aware Contact Specification Language for Ubiquitous Computing,” submitted to FTDCS2003.